# 《编译技术》课程设计文档

## 学号：1406XXXX

## 姓名：XXX

2016年 12 月 28 日

## 一．需求说明

### 1．文法说明

1.＜加法运算符＞ ::= +｜-

定义加减法运算符

例子：

3+4

a+b

3-a

2.＜乘法运算符＞ ::= \*｜/

定义乘除法运算符

例子：

1/2

A\*2

a/b

3.＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

定义关系运算符

例子：

1<2

3<=4

6>5

7>=6

8 != 9

0==1

4.＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

定义字母可以为大小写字母或者下划线

例子：

\_ a B

5.＜数字＞ ::= ０｜＜非零数字＞

定义数字为0~9的单位数字字符

例子：

0 1 7

6.＜非零数字＞ ::= １｜．．．｜９

定义非零数字为1~9的单位数字字符

例子：

1 8

7.＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

定义字符为以单引号开始和结束，其间有且仅有一个字符。

例子：

'+'

'/'

'a'

'8'

8.＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII字符｝"

定义字符串可以为0个，1个，或多个十进制编码为32,33,35-126的ASCII字符

需要注意的是不包括双引号,但是包括空格

字符串的第一个和最后一个字符为双引号

例子：

""//空字符串

"a"//一个字符组成的字符串

"hello, world~"//多个字符组成的字符串

" "//空格组成的字符串

9.＜程序＞ ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞

定义程序。

限定程序的子成分顺序，必须为常量说明在前，然后是变量说明，以上两者均可没有。([]代表0个或一个)

然后是有返回或者无返回的函数定义若干（可以是0个），然后才是主函数 ({}代表0个或多个)

例子:

const int a = 3;//常量说明

int b = 4;//变量说明

void f(int a);//无返回值函数定义

int f2();//有返回值函数定义

void main(){...}//主函数

10.＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

定义常量说明可以是一个或者多个const<常量定义>；

例子：

const int a = 2;

const char c = 'b';

11.＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}| float＜标识符＞＝＜实数＞{,＜标识符＞＝＜实数＞} | char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

定义常量定义可以是整型常量或者字符型常量或者浮点型常量

整型常量的定义格式是int后跟一个或多个＜标识符＞＝＜整数＞的形式，之间用逗号隔开

浮点型常量的定义格式是float后跟一个或多个＜标识符＞＝＜实数＞的形式，之间用逗号隔开

字符型常量的定义格式是char后跟一个或多个＜标识符＞＝＜字符＞的形式，之间用逗号隔开

例子：

int a = 2;

float b = 1.0

char c = 'c', d = 'd';

12.＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝

定义无符号整数的开头必为非零数字，其后可跟若干个数字

该限定表明以0开头的数字串不属于无符号整数

0就不是无符号整数

例子：

1 99

13.＜整数＞ ::= ［＋｜－］＜无符号整数＞｜０

定义整数是以可省略的正负号开头，后跟无符号整数或者0

例子：

0 1 +2 -3

14.＜小数部分＞ ::= ＜数字＞｛＜数字＞｝｜＜空＞

定义小数部分是由一位或多位数字组成，或者为空

例子：

；//代表空

1//1位小数

234//多位小数

15.＜实数＞   ::= ［＋｜－］＜整数＞[.＜小数部分＞]

定义实数是以可省略的正负号开头，后跟整数，再后面可以有小数部分，也可以没有小数部分，小数部分与整数以.为分界

例子：

0

1.

+2

+-3

4.5

16.＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

定义标志符是必须由字母为开头，后跟0到多个字母或者数字的字符串

例子：

a

B1

cc

17.＜声明头部＞ ::= int＜标识符＞ | char＜标识符＞ | float＜标识符＞

定义声明头部

例子：

int a;

char B1;

float cc;

18.＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

定义变量说明为一个或多个变量定义和分号组成的字符串

例子：

int a;

char b1;

float c = 1.0, d = 2.1;

int e[10];

17.＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’){,(＜标识符＞|＜标识符＞‘[’＜无符号整数＞‘]’ )}

定义变量定义，变量可以为一个标志符或者由标志符为起始的数组形式。

例子：

int a2bc\_d

int word[10]

但是不能如下定义：

int 34dfn

int word[-1]

char d[0]

＜可枚举常量＞   ::=  ＜整数＞|＜字符＞

定义可枚举常量，可枚举常量是由整数或者字符组成的

例子：

1 ‘a’

19.＜类型标识符＞ ::= int | char | float

定义类型标识符为整型、字符型和浮点型三种

例子：

int

char

float

20.＜有返回值函数定义＞ ::= ＜声明头部＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’

定义有返回值函数。必须包含头声明头部，参数和复合语句以及必要的括号(大括号，小括号)

例子：

int f(char a){

...

}

char f(char a){

...

}

21.＜无返回值函数定义＞ ::= void＜标识符＞‘(’＜参数＞‘)’ ‘{’＜复合语句＞‘}’

定义无返回值函数。必须包含头声明头部，参数和复合语句以及必要的括号(大括号，小括号)

例子： void f(char a){

...

}

22.＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

定义复合语句是先常量再变量再语句的顺序。

例子：

const int a = 2;

char b = '4';

...//语句列

23.＜参数＞ ::= ＜参数表＞

定义参数

例子：

见参数表处的例子。

24.＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞

参数表由若干个类型标识符和标志符的集合组成，其间用逗号隔开。可以为空。(0-n个)

类型标识符和标志符的顺序不能反。

例子：

int a, char b

int c

//空

注意，int a[2]这种形式是不允许的。

25.＜主函数＞ ::= void main ‘(’ ‘)’ ‘{’＜复合语句＞‘}’

定义主函数，注意括号不能缺失。

例子： void main() {

...

}

26.＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}

表达式是由可省略的+,-开头的若干个由加法运算符连接的项的字符串(1个或多个)

例子：

3\*4+2/6-4\*a

+3

-3+2\*1-c

27.＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

项是由乘法运算符连接的一个或多个因子

例如：

4\*b\*a

1

c\*d

13\*6\*num[15]

28.＜因子＞ ::= ＜标识符＞｜＜标识符＞‘[’＜表达式＞‘]’｜＜整数＞|＜实数＞|＜字符＞｜＜有返回值函数调用语句＞|‘(’＜表达式＞‘)’

因子是由标志符或标志符后跟方括号括起来的表达式或整数或者实数或者字符或者有返回值法的函数调用语句或以圆括号括起来的表达式

例如：

num[15]

num[a]

a

4

(3\*13+5\*a-2/f)

ABC()//ABC是有返回值的函数

29.＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜‘{’＜语句列＞‘}’｜＜有返回值函数调用语句＞;

|＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;｜＜情况语句＞;｜＜返回语句＞;

定义了语句。语句是由上述的几种语句之一构成的

例子：

参照下面的例子即可。

30.＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞‘[’＜表达式＞‘]’=＜表达式＞

定义了赋值语句，注意可以对数组的特定项赋值

例如：

a[3] = 4

b = 5

c = 1\*3

31.＜条件语句＞ ::= if ‘(’＜条件＞‘)’＜语句＞［else＜语句＞］

定义了条件语句，注意条件语句中else可以省略。

例如：

if(a > b){

a = 1;

b = 2;

}else{

b = 3;

}

32.＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞//表达式为0条件为假，否则为真

定义了条件，值得注意的是一个条件中只有一个关系运算符，不支持嵌套

例如：

1!=2 a>=b a 0 1

33.＜循环语句＞ ::= while ‘(’＜条件＞‘)’＜语句＞

定义了循环语句

例子：

while(...){

...

}

其中大括号有时可省略

34.＜情况语句＞  ::=  switch ‘(’＜表达式＞‘)’‘{’＜情况表＞ ‘}’

定义了情况语句

例子：

switch (a)

switch (c+1)

＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}

定义了情况表，情况表是由一个或多个情况子语句构成。

例子：

见下面。

＜情况子语句＞  ::=  case＜可枚举常量＞：＜语句＞

定义了情况子语句，注意冒号不能缺省。

例子：

case 4: b=1

case ‘s’ c=’s’

...

37.＜有返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

定义了有返回值函数调用语句。

例如： f(2, 3)

38.＜无返回值函数调用语句＞ ::= ＜标识符＞‘(’＜值参数表＞‘)’

定义了无返回值函数调用语句。

例子：

同上

39.＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

定义了值参数表，值参数表是若干个（包括0个，即为空）表达式的集合

例子：

//表示空

a

b，c

40.＜语句列＞ ::= ｛＜语句＞｝

语句列是若干（包括0）个连续语句的集合

例子：

{

A=3;

B=2

...

}

41.＜读语句＞ ::= scanf ‘(’＜标识符＞{,＜标识符＞}‘)’

定义读语句是以scanf起始，后接圆括号阔起来的一个或多个以逗号隔开的标识符。

例子: scanf(a, b)

42.＜写语句＞ ::= printf ‘(’＜字符串＞,＜表达式＞‘)’| printf ‘(’＜字符串＞‘)’| printf ‘(’＜表达式＞‘)’

定义写语句是以printf为起始的，后接圆括号括起来的字符串或表达式或者两者都有，若两者都存在，则字符串在先，以逗号隔开。

例子：

printf("%d", a)

printf(a)

printf("Hello, compilier!\n")

43.＜返回语句＞ ::= return[‘(’＜表达式＞‘)’]

返回语句是以return开始，后接可有可无的以圆括号包围的表达式

例子：

return (0)

Return;

### 目标代码说明

类PCODE指令代码不依赖任何具体计算机，其指令集极为简单，指令格式也很单纯，其格式如下：

IMG_256

其中f代表功能码；l表示标志位，用于区分相同功能的不同操作；a的含意对不同的指令有所区别，对存取指令表示位移量，而对其它的指令则分别有不同的含义，见下面对每条指令的解释说明。

lit 0, a : {\*LIT：将常量值取到运行栈顶。a为常数值。\*}

opr l, a : {\*OPR：关系运算和算术运算指令。 l用于区分是否有float类型，a为操作类型。将栈顶和次栈顶的内容进行运算，结果存放在栈顶(switch语句放在栈顶上一位)\*}

lod l, a : {\*LOD：将变量放到栈顶。a为变量在所说明层中的相对位置,l为数组标志位。\*}

sto l, a : {\*STO：将栈顶的内容送入某变量单元中。a,l的含意同LOD指令。\*}

cal l, a : {\*CAL：调用过程的指令。a为被调用过程的目标程序入口地址，l为函数参数个数。\*}

jmp 0, a : {\*JMP：无条件转移指令，a为转向地址。\*}

jpc l, a : {\*JPC：条件转移指令，当栈顶的值为0，转向a的地址，否则顺序执行。l为case标志位\*}

red 0, a : {\*RED：a 读数据并存入地址为a的变量中\*}

wrt l, a : {\*WRT：将栈顶内容或字符串输出，l用来区分输出字符串还是栈顶，a表示打印的字符串在字符串表中的地址\*}

## 二．详细设计

### 1．程序结构

程序入口

词法分析

符号表管理

符号表管理

语法分析

语义分析

生成目标程序

解释执行

### 2．类/方法/函数功能

|  |  |
| --- | --- |
| 函数名 | 作用 |
| 词法分析： |  |
| void getch() | 读取一个字符 |
| int getsym() | 读取一个单词 |
| 语法分析： |  |
| void program() | 程序入口 |
| void constate() | 常量声明 |
| void condef() | 常量定义 |
| void varstate() | 变量声明 |
| void vardef() | 变量定义 |
| void complexsentence() | 复杂语句 |
| void sentencelist() | 语句列 |
| void sentence() | 语句处理 |
| void returnsentence() | 返回语句处理 |
| void condition() | 条件语句处理 |
| void ifsentence() | if语句处理 |
| void switchsentence() | switch语句处理 |
| void loopsentence() | 循环语句处理 |
| void scanfsentence() | 读语句处理 |
| void printfsentence() | 写语句处理 |
| void expression() | 表达式处理 |
| void item() | 项处理 |
| void factor() | 因子处理 |
| 符号表管理： |  |
| void pushnumtable(int address, int kind, int paranum) | 插入数组表 |
| int searchnum(int address) | 返回数组地址 |
| void pushtable(char\* name, int kind, float value, int paranum, int ischar, int isVec) | 插入符号表 |
| int searchst(char\* name, int flag, int needret) | 返回符号表地址 |
| 参数处理： |  |
| void parametertable() | 处理函数声明参数 |
| void insertproc(int paranum) | 插入参数个数 |
| void valueofpara(char\* names) | 处理函数调用参数 |
| 目标代码生成及解释执行： |  |
| void gen(char\* x, int y, float z) | 目标代码生成 |
| void interpret() | 解释执行 |

### 3．调用依赖关系

调用constate()，complexsentence()等

调用getch(),getsym()

程序入口

符号表管理

词法分析

语法，语义分析

调用gen()

错误处理

生成目标代码

调用pushtable()等

解释执行

调用interpret()

调用error()

程序结束

### 4．符号表管理方案

### #define MAX 100

### #define MAXSYM 1000

### typedef struct{

### char name[MAX]; //符号名称

### int kind; //类型(常量,变量,函数,参数)

### /\*

### 0 - CONST//常量

### 1 - VARIABLE//变量

### 2 - FUNCTION//函数

### 3 - PARA//参数

### \*/

### float value; //常量的值

### int datakind; //0->int, 1->char, 2- > float

### int address; //存储这个变量在符号表里的地址

### int paranum; //参数的个数或数组的长度(只有一维数组)

### }basic\_symtable;

### typedef struct {

### basic\_symtable element[MAXSYM];

### int index; //符号表栈顶指针

### int total; //分程序总数

### int indextable[MAXSYM]; //分程序索引数组

### }symtable;

### 存储分配方案

PCODE栈式存储分配见下图：

|  |  |
| --- | --- |
| ... | ... |
| 8 | e |
| 7 | d |
| 6 | F3 |
| 5 | F2 |
| 4 ... | c |
| 3 6 | F1 |
| 2 5 | b |
| 1 3 | a |

符号表 分程序索引表

### 6. 解释执行程序

### void interpret(){

### table.index = 0;

### const int stacksize = 5000;

### int p, t;

### int index = 0;

### float s[stacksize];

### int op[MAX];

### Code i;

### t = 0;

### p = 0;

### while(1){

### i.a = code[p].a;

### strcpy(i.f, code[p].f);

### i.l = code[p].l;

### p++;

### if(strcmp(i.f, "lit") == 0){

### t++;

### s[t] = i.a;

### }

### else if(strcmp(i.f, "lod") == 0){

### t++;

### if(i.l == 0){

### s[t] = maintable.element[(int)i.a].value;

### }

### else{//array

### t--;

### sprintf(result, "%f", s[t]);

### int tempnum = 0;

### while(result[tempnum] != '\0'){

### if(isdigit(result[tempnum]))

### tempnum++;

### else if(result[tempnum] == '.'){

### tempnum++;

### while(result[tempnum] != '\0'){

### if(result[tempnum] == '0')

### tempnum++;

### else{

### cout << "array wrong" << endl;

### return;

### }

### }

### }

### else{

### cout << "array wrong" << endl;

### return;

### }

### }

### if((int)s[t] >= numtable.num[searchnum((int)i.a)].paranum){

### cout << "too long array" << endl;

### exit(0);

### }

### s[t] = numtable.num[searchnum((int)i.a)].f[(int)s[t]];

### }

### }

### else if(strcmp(i.f, "sto") == 0){

### if(i.l == 0){

### maintable.element[(int)i.a].value = s[t];

### }

### else{//array

### sprintf(result, "%f", s[t-1]);

### int tempnum = 0;

### while(result[tempnum] != '\0'){

### if(isdigit(result[tempnum]))

### tempnum++;

### else if(result[tempnum] == '.'){

### tempnum++;

### while(result[tempnum] != '\0'){

### if(result[tempnum] == '0')

### tempnum++;

### else{

### cout << "array wrong" << endl;

### return;

### }

### }

### }

### else{

### cout << "array wrong" << endl;

### return;

### }

### }

### if((int)s[t-1] >= numtable.num[searchnum((int)i.a)].paranum){

### cout << "too long array" << endl;

### exit(0);

### }

### numtable.num[searchnum((int)i.a)].f[(int)s[t-1]] = s[t];

### t--;

### }

### t--;

### }

### else if(strcmp(i.f, "jmp") == 0){

### p = (int)i.a;

### }

### else if(strcmp(i.f, "jpc") == 0){

### if(i.l == 0){//if,while

### if(s[t] == 0){

### p = (int)i.a;

### t--;

### }

### else

### t--;

### }

### else if(i.l == 1){//case

### if(s[t+1] == 0){

### p = (int)i.a;

### }

### }

### }

### else if(strcmp(i.f, "opr") == 0){

### int m = 0;

### switch((int)i.a){

### case 0://return

### p = op[--index];

### table.index--;

### if(index == -1)

### return;

### if(maintable.element[searchst(code[table.num[table.index].address - 1].f, 1, 0)].value == 1){

### if(maintable.element[searchst(code[table.num[table.index].address - 1].f, 1, 0)].datakind != FLOAT)

### s[t] = (int)s[t];

### }

### if(table.num[table.index].para == 0)

### break;

### else{

### while(maintable.element[searchst(code[table.num[table.index].address - 1].f, 1, 0) + m + 1].kind != FUNCTION){

### maintable.element[searchst(code[table.num[table.index].address - 1].f, 1, 0) + m + 1].value = table.num[table.index].tempnum[m];

### m++;

### }

### }

### break;

### case 1:

### s[t] = -s[t];

### break;

### case 2:

### t = t-1;

### if(i.l == 1)

### s[t] = s[t] + s[t+1];

### else

### s[t] = (int)(s[t] + s[t+1]);

### break;

### case 3:

### t = t-1;

### if(i.l == 1)

### s[t] = s[t] - s[t+1];

### else

### s[t] = (int)(s[t] - s[t+1]);

### break;

### case 4:

### t = t-1;

### if(i.l == 1)

### s[t] = s[t] \* s[t+1];

### else

### s[t] = (int)(s[t] \* s[t+1]);

### break;

### case 5:

### t = t-1;

### if(i.l == 1)

### s[t] = s[t] / s[t+1];

### else

### s[t] = (int)(s[t] / s[t+1]);

### break;

### case 8:

### t = t-1;

### s[t] = s[t] == s[t+1];

### break;

### case 9:

### t = t-1;

### s[t] = s[t] != s[t+1];

### break;

### case 10:

### t = t-1;

### s[t] = s[t] < s[t+1];

### break;

### case 11:

### t = t-1;

### s[t] = s[t] >= s[t+1];

### break;

### case 12:

### t = t-1;

### s[t] = s[t] > s[t+1];

### break;

### case 13:

### t = t-1;

### s[t] = s[t] <= s[t+1];

### break;

### case 14:

### t = t-1;

### s[t+1] = s[t] == s[t+1];

### break;

### }

### }

### else if(strcmp(i.f, "red") == 0){

### float tempf;

### int tempi;

### char tempc;

### if(i.l == INT){

### cin >> tempi;

### maintable.element[(int)i.a].value = tempi;

### }

### else if(i.l == CHAR){

### cin >> tempc;

### maintable.element[(int)i.a].value = tempc;

### }

### else if(i.l == FLOAT){

### cin >> tempf;

### maintable.element[(int)i.a].value = tempf;

### }

### }

### else if(strcmp(i.f, "wrt") == 0){

### if(i.l == 0){//string

### cout << str.s[(int)i.a].str;

### }

### else{//expression

### if(i.a == 0){

### cout << s[t];

### }

### else{

### cout << (char)s[t];

### }

### t--;

### }

### }

### else if(strcmp(i.f, "cal") == 0){

### op[index++] = p;

### p = (int)i.a;

### int m = 0;

### if(i.l > 0){

### table.num[table.index].address = (int)i.a;

### table.num[table.index].para = i.l;

### while(maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].kind != FUNCTION){

### table.num[table.index].tempnum[m] = maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].value;

### m++;

### }

### }

### else{

### table.num[table.index].address = (int)i.a;

### table.num[table.index].para = i.l;

### }

### table.index++;

### m = 0;

### int n = i.l-1;

### while(m < i.l){

### if(maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].datakind == FLOAT)

### maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].value = s[t-n];

### else if(maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].datakind == INT)

### maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].value = (int)s[t-n];

### else

### maintable.element[searchst(code[(int)i.a - 1].f, 1, 0) + m + 1].value = (char)s[t-n];

### m++;

### n--;

### }

### t -= i.l;

### }

### else if(strcmp(i.f, "END") == 0){

### return;

### }

### }

### }

### 7.出错处理

出错处理函数void error(int n);

无论n的值为多少，都先打印出错行数及该行内容。

1. n == 0：忽略缺失符号，例如int i 缺失;则报错”lost ;” 同时忽略该错误继续编译
2. n == 1：跳到下一个;之后的单词,例如intt i,m;char c; intt非法，报错”var define wrong”,同时跳读到char继续编译
3. n == 2：跳到下一个) ,例如void abc(intt a) intt非法，报错”type wrong” ,同时跳读到)
4. n == 4：跳到下一个类型标识符,例如int a; const int b；int c；以为常量必须在变量之前定义，所以报错”var define wrong”，跳读到int
5. n == 5：跳到下一个, ; [ 例如int a,a,b[12]; 因为a重复定义，出错，报错”var define wrong”,跳读到,
6. n == 6：跳到下一个;这种错误与n == 1类似，为了防止多次报错。

## 三．操作说明

### 1．运行环境

代码在windows10系统下编写调试，并可在win10及以下版本正常执行。

在CodeBlocks16.01下编写，并可在13.12版本下正确运行

### 操作步骤

编译器运行后将源程序路径输入，按回车开始编译。

编译成功则自动运行，执行目标代码，最后控制台输出结果。

编译中发现错误则继续编译，最后控制台输出错误信息，不执行目标代码。

## 四．测试报告

### 1．测试程序及测试结果

测试程序一：(r1.txt)

考察点：基本计算及选择，输入输出语句

void main()

{

int i,j,k;

printf("please input an int num: ");

scanf(k);

if(k > 1)

i = k + 1;

else

j = k + 2;

if(k > 2)

printf(i);

else

printf(j);

}

输入：1

输出：3END

输入：2

输出：0END

测试程序二：(r2.txt)

考察点：快速排序，主要考察函数调用，参数传递，符号表回填。

int d[10];

void sort(int i, int j){

int x,y,z;

int tmp;

x = i;

y = j;

z = d[(i+j)/2];

while(x <= y){

while(z < d[y]){

y = y - 1;

}

while(z > d[x]){

x = x + 1;

}

if(x <= y){

tmp=d[x];

d[x]=d[y];

d[y]=tmp;

x=x+1;

y=y-1;

}

}

if(x < j)

sort(x,j);

if(i < y)

sort(i,y);

}

void main(){

int i;

i = 0;

d[0] = 4;

d[1] = 7;

d[2] = 1;

d[3] = 3;

d[4] = 2;

d[5] = 6;

d[6] = 5;

d[7] = 9;

d[8] = 0;

d[9] = 8;

sort(0,9);

while(i < 10){

printf(" ", d[i]);

i = i + 1;

}

}

输出： 0 1 2 3 4 5 6 7 8 9END

测试程序三:(r3.txt)

考察点：汉诺塔算法，主要考察函数调用及参数传递，符号表回填。

const int one=1,two=2,three=3;

int num;

void move1(int f,int t)

{ num= num + 1;

printf(" ",f);

printf(" move to ",t);

return ;

}

void moven(int m,int a,int b,int c)

{

if(m==1)

move1(a,c);

else

{

moven(m-1,a,c,b);

move1(a,c);

moven(m-1,b,a,c);

}

return ;

}

void main()

{

int n;

num = 0;

printf("please input the numbers of disks: ");

scanf(n);

printf("the step of moving disks: ",n);

moven(n,one,two,three);

printf(" ", num);

}

输入：3

输出：the step of moving disks: 5 1 move to 3 1 move to 2 3 move to 2 1 move to 3 2 move to 1 2 move to 3 1 move to 3 1 move to 2 3 move to 2 3 move to 1 2 move to 1 3 move to 2 1 move to 3 1 move to 2 3 move to 2 1 move to 3 2 move to 1 2 move to 3 1 move to 3 2 move to 1 3 move to 2 3 move to 1 2 move to 1 2 move to 3 1 move to 3 1 move to 2 3 move to 2 1 move to 3 2 move to 1 2 move to 3 1 move to 3 31END

测试程序四：(r4.txt)

考察点：浮点数开方，考察运算准确性。

float fabs(float x)

{

if(x < 0)

return(-x);

else

return(x);

}

float sqrt(float x)

{

float cur, former;

cur = x;

former = x + x;

while(fabs(former - cur) > 0.00001)

{

former = cur;

cur = (former + x / former)/2;

}

return(cur);

}

void main()

{

float x;

printf("please input a float num");

scanf(x);

printf(sqrt(x));

}

输入：3.14

输出： 1.772END

测试程序五：(r5.txt)

考察点：综合测试，覆盖了全部语法成分。

const int max = 100;

const int min = -0;

const char min\_start = 'a';

const char min\_end = 'z';

const char max\_start = 'A';

const char max\_end = 'Z';

const float pi = 3.14;

const float text = 1.;

int i;

char c;

float f, test;

void error(){

printf("wrong!");

}

int accumulator(int n){

if(n > max){

error();

return (0);

}

else

;

if(n < min){

error();

return (0);

}

if(n == 0){

return (i);

}

else{

i = i + n;

accumulator(n-1);

}

}

char transform(char s){

if(s >= min\_start){

if(s <= min\_end)

s = s - 'a' + 'A';

else{

error();

return ('a');

}

}

else{

if(s >= max\_start){

if(s <= max\_end)

s = s - 'A' + 'a';

else{

error();

return ('0');

}

}

else{

error();

return ('0');

}

}

return (s);

}

void print(int i){

printf(" hello ", i);

}

void main(){

int r;

char temp;

int sum;

char name[11];

float s;

printf("Please input a number:(0,100)");

scanf(r);

if(r > max){

error();

return;

}

if(r < min){

error();

return;

}

printf("Please input your name(11\_char):");

i = 0;

while(i < 11){

scanf(temp);

name[i] = temp;

i = i + 1;

}

i = 0;

if(r != 1)

sum = accumulator(r);

else

sum = r;

s = pi \* r \* r \* text;

f = 2 \* pi \* r;

printf("print answer? Y/N&O: ");

scanf(c);

i = 0;

switch (c){

case 'Y':

while(i < 11){

printf("",transform(name[i]));

i = i + 1;

}

case 'N':

printf("over");

case 'O':

printf("over");

}

printf(" ", s);

printf(" ", f);

printf(" ", sum);

print(max);

}

输入：14 YuxiaoqiANG Y

输出：yUXIAOQIang 615.44 87.92 105 hello 100END

错误程序一：(w1.txt)

考察点：缺失符号

void main()

{

int i,j,k;

scanf(k);

k = k + 1

if(k > 1)

i = k + 1;

else

j = k + 2;

if(k > 2)

printf(i);

else

printf(j);

}

输出:

6

if(k > 1)

lost ;

OVER

错误程序二：(w2.txt)

考察点：重复命名

void main()

{

int i,j,k;

char i;

scanf(k);

if(k > 1)

i = k + 1;

else

j = k + 2;

if(k > 2)

printf(i);

else

printf(j);

}

输出：

4

char i;

same name

OVER

错误程序三：(w3.txt)

考察点：非法返回

void abc(){

return (1);

}

void main()

{

int i,j,k;

scanf(k);

if(k > 1)

i = k + 1;

else

j = k + 2;

if(k > 2)

printf(i);

else

printf(j);

}

输出：

2

return (1);

void can't have return value

OVER

错误程序四：(w4.txt)

考察点：非法调用

int abc(){

return (1);

}

void main()

{

int i,j,k;

scanf(k);

scanf(abc());

if(k > 1)

i = k + 1;

else

j = k + 2;

if(k > 2)

printf(i);

else

printf(j);

}

输出：

8

scanf(abc());

can not scan in a function

OVER

错误程序五：(w5.txt)

考察点：综合报错能力

const int a = 0, aa = 1.0;

const char b = 1, c = 'c';

const float d = 1. ,dd = 'dd';

int f;

char f, g, h;

float i;

void abc(){

return (1);

}

int bcd(){

}

int cde(){

return (1);

}

void main(){

const a1 = 0;

int b1;

int s[10];

b1 = 0;

for(; b1 + 1 < b; b1 = b1 + 1){

printf(b1);

}

scanf(cde());

a = abc();

f = abc() + a;

s[1.0] = 0;

s[3] = s[1] +- 1;

s[4] = s[1] \*/ 2;

while(b1){

s[2] = s[1] + 1

if(s[2] > 10)

b1 = b1 -1;

}

switch(g){

case 'a': {

printf("a");

}

case 'b':{

printf("b");

}

}

}

输入:

输出：

1

const int a = 0, aa = 1.0;

con define wrong

2

const char b = 1, c = 'c';

con define wrong

3

const float d = 1. ,dd = 'dd';

con data type wrong

6

char f, g, h;

same name

10

return (1);

void can't have return value

12

bcd

no return value

18

const a1 = 0;

lost type

22

for(; b1 + 1 < b; b1 = b1 + 1){

for is not legal

25

scanf(cde());

undefined id

26

a = abc();

con can't be assigned

27

f = abc() + a;

no return function

28

s[1.0] = 0;

inside array can't be a float

30

s[4] = s[1] \*/ 2;

fact wrong

33

if(s[2] > 10)

lost ;

OVER

**注：我检查数组是否越界是在运行时检查的，所以在编译过程中没有报错，但是在解释执行的过程中报错，这与gcc编译器是一致的。**

### 2．测试结果分析

测试程序一到五覆盖了全部语法成分，同时设置了多个测试点测试函数递归调用，均未发现问题。

|  |  |
| --- | --- |
| 语法成分 | 覆盖情况 |
| 程序 | 已覆盖 |
| 常量说明 | 已覆盖 |
| 常量定义 | 已覆盖 |
| 变量说明 | 已覆盖 |
| 变量定义 | 已覆盖 |
| 有返回值函数定义 | 已覆盖 |
| 无返回值函数定义 | 已覆盖 |
| 主函数 | 已覆盖 |
| 参数 | 已覆盖 |
| 复合语句 | 已覆盖 |
| 语句列 | 已覆盖 |
| 语句 | 已覆盖 |
| 赋值语句 | 已覆盖 |
| 有返回值函数调用语句 | 已覆盖 |
| 无返回值函数调用语句 | 已覆盖 |
| 条件 | 已覆盖 |
| 条件语句 | 已覆盖 |
| 循环语句 | 已覆盖 |
| 情况语句 | 已覆盖 |
| 情况子语句 | 已覆盖 |
| 读语句 | 已覆盖 |
| 写语句 | 已覆盖 |
| 返回语句 | 已覆盖 |
| 表达式 | 已覆盖 |
| 项 | 已覆盖 |
| 因子 | 已覆盖 |
| 加法运算符 | 已覆盖 |
| 乘法运算符 | 已覆盖 |
| 关系运算符 | 已覆盖 |
| 字符 | 已覆盖 |
| 字符串 | 已覆盖 |
| 实数 | 已覆盖 |

错误程序一到五覆盖了三次测试的所有错误情况，报错位置较准确，错误类型正确。

|  |  |
| --- | --- |
| 错误类型 | 覆盖情况 |
| 类型错误 | 已覆盖 |
| 符号缺失 | 已覆盖 |
| 类型缺失 | 已覆盖 |
| 标识符重复定义 | 已覆盖 |
| void非法返回 | 已覆盖 |
| 不合法标识符 | 已覆盖 |
| 函数错误调用 | 已覆盖 |
| 因子错误 | 已覆盖 |
| 数组错误 | 已覆盖 |

## 五．总结感想

历时近两个月的编译课设到这就暂时告一段落了，通过这两个月的学习，自己完成了一份中等难度的编译器，也对编译的知识有了具体化的应用，感觉挺好的。下面就写一点我对这门课的看法和意见。

编译课设总体来说不是很难，也可能是我选择了中等难度的题目，但是真的做下来觉得知识深度要求不是很高，但是对于细节的把握还是很重要的。例如我再做函数调用的参数处理时，最开始忘记了保存符号表并回填，导致递归调用出现了问题，第一次针对快排的测试就没有通过，后来改好了这个问题三次测试的正确用例就全部通过了。还有就是错误处理这部分，错误处理本身没有一个明确的规定，所以一开始我写的比较简单，在第二次测试的时候被老师指出了不足，回去改正后程序的报错能力也有了提高。

因为目前还没有完成期末考核，所以并不能确定我的程序是否还存在问题，基于目前的测试结果，我觉得我完成的还是挺不错的。这门课没有占用我太多的时间，我觉得只要设计好，代码部分没有什么难的地方。

在这里也向编译课设的课程组提几点我的看法，首先就是希望能有一个自动测试，例如数据结构课程那种自动评判机制，这样不但可以节省老师助教的时间，也有利于我们及时发现问题并改正(不用等到周一或周四)，还有希望可以更加合理的设置分数梯度，我个人觉得，中等难度的课程设计在文法上来看要比高难度的要难，高难度的课程设计主要是考虑优化问题，但是根据学长们的说法，选择高难度的即使不做优化也可以拿到一个很高的分数，我个人觉得不是特别合理，也有可能是学长的信息不准确，总之还是希望能够尽量做到公平就好，了。

最后感谢老师助教们的付出，希望编译原理的课程设计可以越来越好。